Fortran 実習

プログラミング言語 Fortran は、数値計算を主とする自然科学の分野で、現在まで最も多く用いられてきた言語である。本実習ではまず、Fortran を用いた簡単なプログラミング実習を行い、計算機の使い方と Fortran に慣れることを目標としている。

理工系の科学技術計算において、ここ２０年にわたるプログラム言語の主流は FORTRAN77 であった。新しい Fortran 90 では、構造型言語を目指し、ほとんどの文法において抜本的な改良が行われた。にも関わらず、従来の FORTRAN 77 文法を完全に包含しており、77 で書かれたプログラムはそのままコンパイルできる。しかしながら、すでに Fortran の環境はほぼ 90 仕様に移行していることから、本実習ではすべて 90 規格を優先する。

本実習で取り上げるプログラム例は、もとより Fortran 文法を体系的にカバーすることはできないので、さらに実際的なプログラミング演習に対しては、ほかの文法書などを参照していただきたい。以下で取り上げるプログラムは、基本的な Fortran の構文の説明を主に取り上げるため、いずれもアルゴリズムは非常に簡単なものである。しかし、分子軌道法計算や分子動力学法などのある程度大規模なプログラムにおいては、アルゴリズムを的確に把握した上で実地のプログラミングを行うことが非常に重要である。

実習にあたっては、例 7 を除く例題を各自理解したうえでコンピュータ上で走らせてみて、さらにそれらに付随する演習問題のプロクラムをかいてみよ。

*** *********

(1) データの型、宣言、加減乗除、キーボードからの入力、画面上への出力

（例１）２つの整数を読み込んで、その和と差を表示するプログラム。

```
! Sum and Difference
      program exam1
      integer :: i,j,k,l
      READ *, i,j
      k=i+j
      l=i-j
      print *, 'wa= ',k, '      sa=',l
      stop
      end
```

○ Fortran のプログラムの書式には固定形式と自由形式がある。

固定形式（ファイル名の末尾を .f とする）: ７カラム目から７２カラム目までにかき、１行が１つの手続きに対応する。６カラム目が空白でないときは、前の行の継続行と見なされる。１カラム目が C,*,! である行はコメントで、翻訳の際には無視される。

自由形式（ファイル名の末尾は .f９０）: １カラム目から１３２カラム目までにかく。１行に複数の手続きをかくときには；で区切る。行末に＆をつけると、次の行は継続行と見なされる。行中の、! 以下の文はコメントとみなされ、翻訳の際には無視される。

○ プログラム単位　FORTRAN プログラムは１つ以上のプログラム単位からなる。そのうちの１つは主手続き（Main ルーチン）であり、プログラム実行時に最初に呼び出される。その他は外部手続きとなる。例 1 は主手続きのみからなるプログラムである。プログラム単位はその始まりを表す文（この例では program 文）で始まり、end 文で終わる。

○　定数と変数：Fortran のデーターは実行前に値が決まっているもの（定数）と実行時に値が決まるもの（変数）にわけられる。変数名には 先頭が英字で、英字、数字、下線の組み合わせで長さが３１字以内 という約束がある。大文字、小文字の区別はない。

○ データーの型：Fotran の扱うデーターは、 整数型、実数型、倍精度実数型、複素数型、論理型（.true. または .false. の値をとるもの）文字型などいずれかの型をもっている。プログラム単位の中で変数を使う場合その変数がどの型を持っているかを最初に宣言する。これらの宣言が省略されたときには、暗黙のうちに先頭の文字がＡ－ＨとＯ－Ｚの変数は実数型、Ｉ－Ｎの変数は整数型になる。もちろんあらわな宣言のほうが優先される。タイプミスなどによる思わぬバグを防ぐために、暗黙の型宣言を無効にする文 implicit none をプログラム単位の最初に置いて全ての変数をあらわに定義することも多い。

○ 入出力文：入出力文を使って、キーボードまたは事前に準備したファイルから変数に値を読み込むことができる。また画面上やファイルにデーターの値を出力する事ができる。

○ 代入文：Fortran では a=b としたとき、b の値を変数 a に代入するという意味になる。

○ 演算：加減乗除 $+, -, *, /$ やべき乗 $**$ を使った数値計算や .and. .or. など使った論理演算を行うことができる。かっこが無いときの演算順序の優先順位は $** > *, / > +, -$ となる。同じ優先順位のものが並んでいる場合には、左から順に処理される。例えば、$a/b*c$ は $(a/b)*c$, また、 $a**b**c$ は $(a**b)**c$ と理解される。自分でみて曖昧なときはかっこをつけておくのが無難。なお、整数型同士の除算では結果も整数となり、余りが切り捨てられることに注意。

例１の場合（このプログラムは固定形式で書かれている）

１行目　コメント行

２行目　このプログラムの名前は exam1 である。

３行目　変数 i ～ l は整数型である。

４行目 変数 i, j の値をキーボードから読み込む。（read(5,*) i, j と書いても良い。）

５、６行目　i+j を計算し変数 k に代入する。i-j を計算し変数 l に代入する。

７行目　変数 K, l の値を画面に出力する。（write(6,*) 'wa=',k と書いても良い。）

８行目　実行を終了する。（stop 文が無くても Main ルーチンの end 文に到達すれば実行は終わるが、条件判断を伴う分岐があって、ある分岐に対してのみ実行を終了させたいときに使われることが多い）

９行目　プログラム単位の終わり

（演習１）２つの実数を読み込んで、その加減乗除すべての計算結果を出力するプログラムを書いてみよ。またそれを実行せよ。

（演習２）元金 a を年利率 r% の複利でｎ年間預けたときの全利子を計算せよ。

** ********* œ(I44

2) 組み込み関数

（例２）任意の角度（単位 degree）を読み込んで、その sin と cos を表示するプログラム

```
!  SIN and COS with angular transformation
      program exam2
      real :: x,pi,sx,cx,pi
      pi=3.14159265
      print *, ' INPUT X / DEGREE'
      read *, x
      x=x*pi/180.0
```

```
sx=sin(x)
cx=cos(x)
print *, 'SIN X = ',sx
print *, 'COS X = ',cx
end
```

○ 組み込み関数： Fortran には 3 角関数、指数関数、対数関数、平方根などの簡単な関数が組み込み関数として用意されている。組み込み関数はかっこ内に引数を伴って用いられる。（たとえば　A=EXP(-2.0), I=MAX(5,2,-1)　など。）下に一部の組み込み関数とその簡単な説明を与える。

表 1: 組み込み関数の例　　[$XXX(a)$ または $XXX(a_1, a_2, \ldots)$]

関数名	定義		引数の個数	引数の型	関数の型
SIN	正弦	$\sin a$	1	実	実
COS	余弦	$\cos a$	1	実	実
ACOS	逆余弦	$\cos^{-1} a$	1	実	実
EXP	指数	e^a	1	実	実
LOG	自然対数	$\ln a$	1	実	実
SQRT	平方根	$\sqrt{a}, a \geq 0$	1	実	実
ABS	絶対値	$\|a\|$	1	実	実
MOD	剰余	$a_1 - \mathrm{int}(a_1/a_2) * a_2$	2	整・実	引数と同じ
MAX	最大値	$\max(a_1, a_2, \ldots)$	2 以上	整・実	引数と同じ
MIN	最小値	$\min(a_1, a_2, \ldots)$	2 以上	整・実	引数と同じ
INT	整数型への変換		1	整・実	整
REAL	実数型への変換		1	整・実	実

○ 文関数：組み込み関数にない関数を自分で定義することもできる。その関数が 1 行の式で書けるときには、定義式を実行文の前に書いておけばよい。

（例） fermi(beta,energy)=1.0/(1.0+exp(-beta*energy))

関数の定義に複数の文が必要なときや、いくつものプログラム単位で同じ関数を利用したいときには、後にでてくる関数副プログラムを使う。

（演習３）2 次方程式 $ax^2 + bx + c = 0$ の係数 a, b, c を読み込んで、その根を表示する。（ここではとりあえず実数解しか持たない場合だけ考えればよい。）

（演習４） 三角形の 3 辺の長さ x, y, z を与えたとき、それぞれに対する角度 (degree) を求める。（ここではとりあえず入力される x, y, z で必ず三角形ができるものとする。）

（演習５） ある 1 日の中で、a 時 b 分から c 時 d 分（２４時間制）まで何時間何分あるかを求める。結果の表示も、文字定数を用いて分かりやすくせよ。

** *********

3 丸め誤差、桁落ち

計算機で扱える実数は近似的な値であるために計算によっては丸め誤差、桁落ちの誤差などが深刻になることがある。例えば（演習３で）2 次方程式の解の公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

をそのままの形で使うと $|b|$ と $\sqrt{b^2-4ac}$ がきわめて近いときには桁落ちによって絶対値の小さな解の（相対）誤差が大きくなる。このようなことをなるべくさけるため、

$b>0$ の場合　$x_1=\dfrac{-b-\sqrt{b^2-4ac}}{2a}, x_2=c/(ax_1)$,　　$b<0$ の場合　$x_2=\dfrac{-b+\sqrt{b^2-4ac}}{2a}, x_2=c/(ax_1)$

を用いる。

（演習６）２次方程式 $x^2-2px+q=0,\ p>0$　（q は非常に小さな数）の解を通常の解の公式を用いて計算した場合と上で述べたように桁落ちをさけた形で計算した場合とで結果を比較せよ。

*** *********

４ 条件判断による処理の流れの制御

○条件判定を行って判定結果ごとに処理をかえるときには、IF 文や CASE 文を用いる。

○ブロック **IF** 文

```
IF(条件 1) THEN
    処理（１）
ELSE IF (条件２) THEN
    処理（2）
ELSE
    処理（3）
END IF
```

条件１が成り立つとき処理（１）を行い、成り立たないときには条件２が成り立つかどうか判定する。そのとき条件２成り立てば処理（２）を行う。条件１も条件２も成り立たないとき処理（３）を行う。ELSE IF 文と処理（２）および、ELSE 文と処理３はなくてもよい。処理（１）と処理（２）は複数の文でかまわない。

○IF(条件)　文（単一の文）

の形式もある。（論理ＩＦ文）

これは、

```
IF（条件）THEN
  文
END IF
```

とおなじである。

○条件判定には論理型変数または、論理式が入る。論理式は、多くの場合２つの数の大小関係の比較が用いられる。

>(.GT.), >=(.GE.), <(.LT.), <=(.LE.), ==(.EQ.), / =(.NE.)

（例３）任意の２つの自然数を読み込んで、ユークリッドの互助法を用いて最大公約数を求める。

```
!  G.C.M with Euclid
     program exam3
     integer :: i,j,k
     read *, i,j
     if(j>i) then
       k = j
       j = i
       i = k
     end if
     if(j <= 0) stop
 100 continue
     k = mod(i,j)
     if (k == 0) goto 200
     i = j
     j = k
     goto 100
 200 PRINT *, 'G.C.M. = ',j
     stop
     end
```

◦ ＧＯ ＴＯ文と文番号 — ＧＯ ＴＯ文はいうまでもなく、与えられた文番号に飛んで次の処理を行うものである。これを多用すると処理の流れが見通し悪くなるため、いわゆる構造化プログラミングにとって望ましくないものとされるが、実際にはまだよく使われている。

文番号は、先頭に５桁までの整数で書く。通常の文には文番号は必要ないが、ＧＯ ＴＯ文の行き先にはなくてはならない。この例で文番号１００のついているＣＯＮＴＩＮＵＥ文は、何もしないという働きがあり、文番号をもったラベルとして用いることが多い。

◦ なお、算術ＩＦ文と呼ばれる形式もある。

if（整数または実数）n_1, n_2, n_3

これは、(　　) 内が負、０、正それぞれの場合に、文番号 n_1, n_2, n_3 にジャンプするものである。

◦ インデントと空白 — この例ではブロックＩＦ文の中で字下げをしている。字下げ（インデント）は計算機の翻訳にとっては意味をもたないが、人間がプログラムを読みやすくするために適当に用いるとよい。単語のすき間にある空白も同様である。

（演習７）任意の正の整数を読み込んで、偶数か奇数かを判定せよ。

（演習８）前回の演習 3 で、2 次方程式の解が虚根になるときは、そのことを述べて終るように改良せよ。桁落ちの問題のないプログラムになっていればなおよろしい。

（演習９）自然対数の底ｅを、精度 10^{-5} で求めよ。（公式

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

を用い、足される項が 10^{-5} よりも小さくなったときに処理をうち切る。加える項は、一つ前の項から計算することができることに注意せよ。)

*** *********

5 処理の繰り返し

○ **DO** 文

ある処理を繰り返して行いたいときには DO 文を用いる。

```
(ラベル:)    do = i1, i2

               [ 処理 ]

             end do (ラベル)
```

このとき、カウンタ I (別に I でなくても整数ならよい) が、I1 から I2 まで 1 ずつ増えて処理を繰り返す。見やすくするため、ラベルをつけてもよい。

○ ＤＯ文は、いれ子にして用いることもできる。

```
(out:)       do i = i1, i2
               [        ]
(in:)          do j = j1, j2
               [        ]
               end do (in)
               [        ]
             end do (out)
```

また、ＤＯ文のカウンタは必ずしも１ずつ増えなくてもよい。

do i= 初期値, 終値, 増分

３番目の増分は、省略すれば１になる。

繰り返しの回数があらかじめ決まっておらず、何かの条件が成立したときに途中で繰り返しを終了したい場合は

```
do
  処理
  if(条件) exit
  処理
end do
```

ループの先頭でループを抜け出すかどうかを判定するときにはさらに、

```
do while(論理式)
  処理
end do
```

という書き方ができる。このとき論理式が真である限り処理が繰り返される。(演習 9 にこれを用いるとプログラムがすっきりする。) (例４) N個の実数を１つずつ読み込んで、最大のものと最小のものを求める。

```
 !  Maximum and Minimum
      program exam4
      read *, n
      do i=1,n
        read *, x
        if(i == 1) THEN
          xmin = x
          xmax = x
        end if
        if (x > xmax) xmax=x
        if (x < xmin) xmin=x
      end do
      print *, 'MAXIMUM= ',xmax
      print *, 'MINIMUM= ',xmin
      stop
      end
```

○ この例では、暗黙の型宣言を使っている。

(演習１０) 10^{-8} を (単精度) 実数型および倍精度実数型で 10^n 回 $(n = 1 \sim 8)$ 足した場合の結果を比較せよ。

(演習１１)

$$S = \int_0^1 \frac{1}{1+x^2} dx$$

を台形公式で求める。(正確な解は $\tan^{-1} 1$ となる。) 分点の個数Nを読み込み、そのときの積分の値を表示せよ。分点は等間隔に選んでよい。また、N の値を増やしていくと、積分値はどのように変化していくか観察せよ。

(演習１２) ある自然数に対して、自分自身を除くすべての約数の和がもとの数に等しいとき、それを完全数という。N を適当に与えて、2 から N までの自然数のなかにある完全数をすべて求めよ。

*** *********

配列、入出力フォーマット

(例５) N個の粒子の質量とデカルト座標を与えて、その重心と重心まわりの慣性モーメントテンソルを求める。

```
!  CENTER OF MASS AND INERTIA MOMENT TENSOR
!
      PROGRAM EXAM5
      IMPLICIT REAL(8) (A-H,O-Z)
   PARAMETER(NMAX=20)
   DIMENSION :: W(NMAX),R(NMAX,3),RW(NMAX,3),R0(3),WRR(3,3),G(3,3)
```

```
      DATA ZERO/ 0.0D+00/
!
!     INPUT
!
      READ *, N
      DO I=1,N
        READ *, W(I),(R(I,K),K=1,3)
      END DO
!
!     CENTER OF MASS
!
      TOTW = ZERO
      DO K=1,3
      DO I=1,N
        RW(I,K)=R(I,K)*W(I)
      ENDDO
      ENDDO
      TOTW=ZERO
      DO I=1,N
        TOTW=TOTW+W(I)
      ENDDO
      DO K=1,3
        R0(K)=ZERO
      ENDDO
      DO K=1,3
      DO I=1,N
        R0(K)=R0(K)+RW(I,K)
      ENDDO
      ENDDO
      DO K=1,3
        R0(K)=R0(K)/TOTW
      ENDDO
!
!  MOMENT OF INTERITA
!
      DO K=1,3
      DO I=1,N
        R(I,K)=R(I,K)-R0(K)
      ENDDO
      ENDDO
      DO K1=1,3
      DO k2=K1,3
        WRR(K1,K2)=ZERO
        DO I=1,N
```

```
            WRR(K1,K2)=WRR(K1,K2)+RW(I,K1)*R(I,K2)
          ENDDO
        ENDDO
      ENDDO
      G(1,1)=WRR(2,2)+WRR(3,3)
      G(2,1)=-WRR(1,2)
      G(3,1)=-WRR(1,3)
      G(1,2)=-WRR(1,2)
      G(2,2)=WRR(1,1)+WRR(3,3)
      G(3,2)=-WRR(2,3)
      G(1,3)=-WRR(1,3)
      G(2,3)=-WRR(2,3)
      G(3,3)=WRR(1,1)+WRR(2,2)
!
!     OUTPUT
!
      WRITE(6,9010)
 9010 FORMAT(/,5X,14HCENTER OF MASS)
      WRITE(6,9020) X0,Y0,Z0
 9020 FORMAT(5X,'X=',F15.7,2X,'Y=',F15.7,2X,'Z=',F15.7)
      WRITE(6,9030)
 9030 FORMAT(/,5X,21HINERTIA MOMENT TENSOR)
 9040 FORMAT(10X,3F15.7)
   DO K1=1,3
     WRITE(6,9040) (G(K1,K2),K2=1,3)
   ENDDO
      STOP
      END
```

○このあたりからやや実際的なプログラム例を取り上げる。この例は今までよりはやや長いが、行っていることは簡単で、N個の粒子の質量とX，Y，Z座標を W_i, X_i, Y_i, Z_i とするとき、重心

$$(X_0, Y_0, Z_0) = (\frac{\sum_i W_i X_i}{\sum_i W_i}, \frac{\sum_i W_i Y_i}{\sum_i W_i}, \frac{\sum_i W_i Z_i}{\sum_i W_i})$$

とそのまわりの慣性モーメントテンソル

$$\left(\begin{array}{ccc} \sum_i W_i (Y_i'^2 + Z_i'^2) & -\sum_i W_i X_i' Y_i' & -\sum_i W_i X_i' Z_i' \\ -\sum_i W_i Y_i' X_i' & \sum_i W_i (Z_i'^2 + X_i'^2) & -\sum_i W_i Y_i' Z_i' \\ -\sum_i W_i Z_i' X_i' & -\sum_i W_i Z_i' Y_i' & \sum_i W_i (X_i'^2 + Y_i'^2) \end{array} \right)$$

$$\begin{aligned} X_i' &= X_i - X_0 \\ Y_i' &= Y_i - Y_0 \\ Z_i' &= Z_i - Z_0 \end{aligned}$$

を求めているだけである。

○質量と座標はN個の粒子をまとめて、配列変数（ W(I),I=1,N など ）として扱う。そうすればプログラムも見やすく、Nが変化したときの対応も容易である。用いる変数が配列であることを、ＤＩＭＥＮＳ

ＩＯＮ文などで宣言する。(　)内の数は配列変数の要素の数で、ここでは２０個分を用意している。(Ｎ＞２０のときには、PARAMETER 文中の数字をもっと大きな数にしなければならない。)

○ **DIMENSION 文**：ＤＩＭＥＮＳＩＯＮ文で、V(下限：上限)と２つの添え字を指定すれば、配列 V の添え字の下限と上限を与えられる。上の例のように１つしか添え字を指定しなければ、下限は１になる。型と配列を同時に指定することもできる。また、配列変数は２次元以上でも使用できる。

```
real, dimension(1:100) :: a,b,c
real(8) :: a(100),b(100),c(0:100)
real(8) :: d(100,100)
```

○ 配列の演算: 配列の演算(例えば行列の足し算、かけ算など)は配列要素に適当な順でアクセスして処理を行うことになるが、Fortran90 では、この処理をコンパイラにまかせることができる。たとえば、X,Y を同じ大きさの配列として、

```
z_sub=1.0
z_mul=2.0*x
z_add=x+y
z_sub=x-y
z_mul=x*y
z_div=x/y
```

とすると全ての配列要素に同じ大きさのスカラー量を代入したりかけたり, 対応する要素間の加減乗除が行える。

配列を引数にした組み込み関数も用意されていてベクトルの内積、行列のかけ算などは、

```
real(8) :: a(100),b(100),x(10,8),y(8,6),z(10,6)
s=cdot(a,b)
z=matmul(x,y)
```

などとかくことができる。X*Y と matmul(X,Y) は異なることに注意。さらに、配列の一部分をあたかも一つの配列とみなして操作することもできる。

○ **FORMAT 文** 入出力の形式を計算機にまかせるなら、read(5,*) write(6,*)（または read *,　print *,）と書けばよいが、書式をきちんと揃えるように指定したい場合がある。このとき、READ(5,9000) や WRITE(6,9000) のように２番目のパラメーターに＊ではなく文番号を記し、その文番号のＦＯＲＭＡＴ文をおく。ＦＯＲＭＡＴ文はＥＮＤの前ならばどこに置いてもよい。(Fortran90 では、FORMAT の内容を直接 '　' でくくって文番号を入れる位置に write(6,'(2I5,3F,F10.3)') の様にかくこともできる。print 文の場合には print '(2I5,3F,F10.3)', とする。)ＦＯＲＭＡＴ文に用いる書式設定には多くの種類があってすべて説明しきれないので、ここでは主なものだけ例をあげて説明する。よく用いられるのは、Ｉ型、Ｆ型、Ｅ型、Ｄ型、Ｘ型、Ｈ型、／型などの編集記述子である。

• FORMAT(2I5,3X,F10.3)

```
␣␣␣15␣␣-32␣␣␣␣␣-125.031
```

Ｉ型	In	ｎカラム分用いて整数型の数を右詰めで表示する。
Ｘ型		１カラム分スキップする。

Ｆ型　Fn.m　全体でｎカラム分、うち小数点以下mカラム分の実数型の数を
右詰めで表示する。

なお、編集記述子の前に整数がつくと、その回数分だけ繰り返す。

- FORMAT(3X,12HTEMPERATURE=,F10.2,/,3X,8HDENSITY=,D15.7)

```
␣␣␣TEMPERATURE=␣␣␣␣300.00
␣␣␣DENSITY=␣␣0.9832149D+00
```

Ｈ型　nHœ(I4ldotsＨ型編集子に続くｎカラム分をそのまま表示する。

／型　改行する。

Ｅ型　En.m　全体でｎカラム分、有効数字mカラム分の浮動小数点表示。
0.98œ(I4rm E-01 = 0.098
0.98œ(I4rm E+01 = 9.8
なお、倍精度実数型のときは、Ｅ型のかわりにＤ型をつかう。

（演習１３）N 個の点電荷のそれぞれの電荷と３次元座標が与えられたとき、必要ないくつかの点での電位と電場ベクトルを計算するプログラム。

（演習１４）ある試験の N 人の点数を読み込み、全体の平均点と分散を求め、さらに各人の偏差値を求めよ。

** *********

外部手続き、配列の動的割付

（例６）M行N列の行列を用意し、それを整形して表示するプログラム

```
!  MATRIX PRINT
!
      program exam6
      real, allocatable :: a(:,:)
!
      print *, 'Matrix Size M x N'
      read *,   m, n
!
      allocate( a(m,n) )
!
      call matinp(a,m,n)
      call matout(a,m,n)
      stop
      end
!
      subroutine matinp(x,m,n)
```

```
      dimension x(m,n)
!
!     UNIT MATRIX FOR EXAMPLE
!
      do i=1,m
      do j=1,n
        x(i,j)=real(i+j)
      end do
      end do
      return
      end
!
      subroutine matout(x,m,n)
      dimension x(m,n)
      imax = 0
      print *, ' *** PRINT OUT A MATRIX ***'
   10 imin = imax+1
      imax = imax+8
      if(imax > n) imax=n
      write(6,9010) (j,j=imin,imax)
      DO i=1,m
        write(6,9020) i,(x(i,j),j=imin,imax)
      end do
      if(imax < n) go to 10
      return
 9010 format(/,5X,8I9)
 9020 format(I5,8F9.4)
      end
```

◦ 配列の動的割付け: 以前の例では、配列の大きさがあらかじめ決まっていなかったため、十分大きなサイズを確保しておいた。Fortran90 では、必要十分な大きさの配列を実行時に確保する事ができる。

*動的割付け

型、ALLOCATABLE :: 配列変数名 (:)

....

ALLOCATE (配列変数 (サイズ))

*割付け解除

DEALLOCATE (配列変数)

◦ **SUBROUTINE** 文と **CALL** 文 計算機に行わせる一連の処理の中で、ある程度まとまった機能を持

つ処理のかたまりをサブルーチンとして独立させると便利なことが多い。

$$\text{CALL}\boxed{\text{XXX}} \Longleftrightarrow \begin{cases} \text{SUBROUTINE}\boxed{\text{XXX}} \\ \vdots \\ \text{RETURN} \\ \text{END} \end{cases}$$

主プログラムの中でＣＡＬＬ文に出会うと、そこから指定された名前のサブルーチンの実行に移り、ＲＥＴＵＲＮ文で元のＣＡＬＬ文に戻る。この例では２つのサブルーチン MATINP と MATOUT が用いられ、それぞれ行列の入力と出力の機能を担っている。サブルーチンを適当に使うことによってプログラムの内容が見やすくなるばかりでなく、同じサブルーチンをいろいろな場面でＣＡＬＬして用いれば、プログラム自体も効率的になる。

サブルーチンの呼び出しに際しては、引数の受渡しができる。

CALL	MATINP	(A,	M,	N)
	$\Updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$
SUBROUTINE	MATINP	(X,	M,	N)

それぞれ左から順に、同じ型の変数として対応がつく（この例では、A–X. . . 実数型配列　M–M, N–N . . . 整数型）。呼び出す側での引数を実引数とよび、呼び出される側での引数を仮引数と呼ぶ。ここで受け渡しの変数の型が一致していないと、正しい対応が得られない。一般にサブルーチンは引数を受け取り、それに適当な処理をほどこして送り返すことが多い。引数のない SUBROUTINE も許される。

◦ なお、関数の定義を外部手続きで行う時は関数副プログラムを使うことも多い。

```
! 主プログラム
      PROGRAM MAIN
      ⋮
      A = FUNC(A1, A2, . . . )
      ⋮
      END
! 関数
      REAL FUNCTION FUNC*8(A1, A2, . . . )
      ⋮
      FUNC = [計算結果]
      END
```

これは SUBROUTINE を用いれば、下のようになる。

```
! 主プログラム
      PROGRAM MAIN
      ⋮
      CALL SUB(A, A1, A2, . . . )
      ⋮
      END
! サブルーチン
```

```
SUBROUTINE SUB(A, A1, A2, ...)
REAL*8 A
⋮
A = 計算結果
RETURN
END
```

両者の形式の違いと共通性は、上の２つの例から明らかである。例２で述べた組み込み関数も、やはり関数の一種である。

（演習１５）N次多項式関数 f(x)

$$f(x) = \sum_{j=0}^{N} a_j x^j$$

の次数 N と係数 $a_0 \sim a_N$ を与えたとき、方程式 $f(x) = 0$ の解の１つを Newton 法で求めるプログラムを作れ。 Newton 法の初期値は $x = 0$ 、収束精度は 10^{-10} とし、計算は倍精度実数で行え。

(演習１６) 行列 A と行列 B を与えたとき、その積の行列 $C = A \times B$ を求めるサブルーチンをつくれ。ただし、行列積の組込み関数 MATMUL は使わないこと。行列 A,B の次元はそれぞれ L 行 M 列、M 行 N 列とし、それらも引数として与えるようにせよ。(すなわち、SUBROUTINE $\boxed{***}$(A,B,C,L,M,N) で始まるプログラムをかくことになる。) また、できあがったサブルーチンをテストするために、そのサブルーチンを呼び出す簡単な主プログラムも用意せよ。

** *********

ファイルの入出力

（例７）ファイルの入出力

今までのすべての例は、外部とのデータの入出力がキーボード（入力）とコンソール（出力）であった。もう少し大規模なプログラムでは、その他のファイルとの入出力を行うことが多い。例えば、分子動力学計算でトラジェクトリーの計算結果をファイルに出力して保存しておき、後でそれを読み込んで解析する。または ab initio 分子軌道計算では、膨大な二電子積分をメモリー上に保存することが不可能なので、分割してファイルに書き出しておき、ＳＣＦ計算の段階でそれを読み込んできて利用する。それでは、分子動力学計算の最も大まかな骨組みの例を示す。(この例は、呼び出すサブルーチンがないので、そのままでは走らない。)

```
!  MOLECULAR DYNAMICS
!
      PROGRAM EXAM7
      MODULE variables
       INTEGER(4) :: N,MAXSTP
       REAL(8)    :: DT
!        N -------- 粒子の総数
!        DT ------- 時間の間隔
!        MAXSTP --- 全体の時間ステップ数
       REAL(8)    :: RX(30),RY(30),RZ(30),VX(30),VY(30),VZ(30)
```

```
       REAL(8)     :: FX(30),FY(30),FZ(30)
!       RX,RY,RZ --- 粒子の位置（x,y,z 成分）
!       VX,VY,VZ --- 粒子の速度
!       FX,FY,FZ --- 粒子にかかる力
!
      END MODULE
!
!  MAIN
      USE variables
!
      DATA KOUT/ 1/
!
      CALL INIT
!        初期条件の用意
      OPEN(KOUT,FILE='MDOUT',STATUS='UNKNOWN',ACCESS='SEQUENTIAL',
     &          FORM='UNFORMATTED')
      TIME = 0.0D0
      DO I=1,MAXSTP
      CALL FORCE
!        分子間力の計算
      TIME = TIME + DT
      CALL FORWRD
!        １ステップ分の時間発展
      WRITE(KOUT) TIME,RX,RY,RZ,VX,VY,VZ
      END DO
      CLOSE(KOUT,STATUS='KEEP')
      STOP
      END
```

このプログラム例は分子動力学計算の最も大まかな骨組みで、サブルーチン内にはここでは立ち入らない。

◦ ＯＰＥＮ文・ＣＬＯＳＥ文： 標準入出力以外のファイルとの入出力を始める前には、ＯＰＥＮ文をつけ、入出力がすべて終わったらＣＬＯＳＥ文を書く。 OPEN されたファイルには装置指定子 (この例では KOUT=1) がつけられる。この数は５番と６番以外の自然数なら何でもよい。５番は標準入力、６番は標準出力にあらかじめ決められている。複数のファイルを同時に OPEN された状態にしておくには、それぞれ異なった番号を割り当てればよい。 ＯＰＥＮ文とＣＬＯＳＥ文の各項目を下に示す。

* OPEN 文

FILE — ディスク上のファイル名
STATUS — ∘'NEW' — 新しく作るファイル
∘'OLD' — 既に存在するファイル
∘'UNKNOWN' — 既に存在すればそれをＯＰＥＮし、
なければ新たに作る。
省略時は 'UNKNOWN'
ACCESS — ∘'SEQUENTIAL' — 逐次アクセス型
∘'DIRECT' — ダイレクトアクセス型
（今はダイレクトアクセス型の使い方を説明しない。）
省略時は 'SEQUENTIAL'
FORM — ∘'FORMATTED'
∘'UNFORMATTED'

今までの標準入出力はすべて FORMATTED であった。これは READ(5,*) や READ(5,9000) のようにフォーマット指定子（*, 9000, etc）をつけて入出力する。UNFORMATTED のファイルとの入出力に対しては、READ(1) のようにフォーマット指定子をつけない。UNFORMATTED のファイルは、そのまま表示しても人間が読むことが出来ない（バイナリーファイル）が、ファイル自体がコンパクトにすむという利点がある。

* CLOSE 文
STATUS — ∘'KEEP' — ファイルを保存する。
∘'DELETE' — CLOSE 時にファイルを消去する。
省略時は 'KEEP'

∘ ここで作られたファイルを後で読み込んで使用したければ、

```
      ⋮
      DATA KIN/ 1/
      OPEN(KIN,FILE='MDOUT',STATUS='OLD',ACCESS='SEQUENTIAL',
     1FORM='UNFORMATTED')
      ⋮
      READ(KIN,END=2000) TIME,RX,RY,RZ,VX,VY,VZ
      ⋮
2000  CLOSE(KIN,STATUS='KEEP')
      ⋮
```

のようにすればよい。READ 文についた END=$\boxed{\text{文番号}}$ は、読み込みがファイルの最後まで達したときにジャンプする文番号を示す。

∘ **MODULE** 文: 主プログラムやサブルーチンなどのプログラム単位をまたがった場合、たとえ同じ名前の変数であっても本来は互いに関係のない別物であり、引数で引き渡しする変数のみが共有される。しかし、場合によってはそれでは不便なことがある。例えば物理定数やパラメーターなどを複数のプログラム単位で共有させたいとき、いちいちサブルーチンの引数としてやり取りしていたのではごちゃごちゃしてしまう。このようなとき、Fortran90 では、独立したプログラム単位である「モジュール」に、共有する変数や定数、配列をまとめて書いておき、一回の宣言でそれを使うことができる。

*モジュール副プログラム

MODULE モジュール名

–共有変数などの型宣言–

…………

END MODULE

*モジュール使用宣言（サブルーチンの先頭で）

USE モジュール名

○ **COMMON 文**：COMMON 文を用いて変数を共有することもできる。ただし、COMMON 文では、同じコモンブロックのデータであれば、異なるサブルーチンで違う名前の変数を用いてもよい。

```
SUBROUTINE A
IMPLICIT REAL*8 (A-H,O-Z)
COMMON/XXX/X(30),IM,JM
⋮
SUBROUTINE B
IMPLICIT REAL*8 (A-H,O-Z)
COMMON/XXX/X(30),IM,JM
⋮
```

この例のように、異なるプログラム単位間で同じコモンブロック名（ XXX ）をもつ COMMON 文が宣言されると、その変数（倍精度実数の配列変数 X(30)、及び整数変数 IM,JM ）は共有される。（COMMON 文で配列変数を宣言するときには DIMENSION 文は必要ない。）COMMON の指定があるときには、一つのプログラム単位でその変数を変更すると、別のプログラム単位に実行が移ってもその変数の変更後の値が保持されている。その際、思いがけない副作用がないように気をつけるべきである。

（演習１７）上の例題でつくられたファイル MDOUT がすでに得られていたとする。そのとき、すべての時間ステップのデータを集計して、粒子の運動エネルギー分布のヒストグラムをもとめるプログラムを作成せよ。ただし、i 番目の粒子の運動エネルギー $e_k(i)$ は

$$e_k(i) = \frac{1}{2}w[v_x(i)^2 + v_y(i)^2 + v_z(i)^2]$$

で与えられる。w は粒子の質量、$v_x(i), v_y(i), v_z(i)$ は i 番目の粒子の速度の x, y, z 成分である。ヒストグラムのエネルギー幅は H とせよ。

** *********

（例８）乱数の使用

```
program example8
integer seed(1:1)
print *, 'number of random_number'
read *, n
print *, 'seed'
```

```
read *, seed(1)
call random_seed(put=seed)
do i=1,n
  call random_number(x)
  print *, x
enddo
end
```

∘ 乱数 Fortran 規格にはないが、ほとんどの処理系で 0 から 1 までの（擬似的な）一様乱数を発生させる組み込みルーチンが用意されている。大概の場合、乱数の種となる整数を与えた後、乱数発生の組み込みルーチンを繰り返し call する事で一連の実数乱数を発生させる形になっている。

（演習１８）半径 1 の円の面積を乱数を用いたモンテカルロ法でもとめよ。（半径 1 の円の面積は、0 から 1 までの一様乱数 x、y を発生させたときに 点 (x,y) と原点の距離が 1 以下となる確率の 4 倍に等しい。モンテカルロの試行回数を 10^2, 10^4, 10^6 としたものををそれぞれ３０回行い求められた円の面積の分布を調べてみよ。）
（演習１９）1 次元の直線上を原点から出発して歩幅 w=1.0 で n 歩く。ただし、それぞれの 1 歩は確率 1/2 で右または左に進むとする。このランダムウオークの試行を 1000 回繰り返したとき、歩き終わった地点と原点の距離の 2 乗平均を n に対してプロットせよ。

** *********

このほか Fortran90 には C 言語と同様に再起呼び出し、構造型、ポインタなどの概念があり、必要に応じて利用されるが、本演習の範囲を超えるのでここでは述べない。

以上およそプログラミングの概略を紹介した。これまでの知識があれば、実際のプログラムをかなりの程度まで読みすすむこともできるはずである。その他の知識が実際に必要になったときには、文法書を参照してほしい。